First Lasso experiments

Set.seed was set to 4 to make results reproducable

prerun this please: code\_below is full gitcode from ’heiko\_wolfgang\_code\_test.R"

{  
 {  
library(magrittr)  
library(dplyr)  
library(glmnet)  
library(purrr)  
library(tidyverse)  
#?glmnet  
library(coefplot) #for extracing non 0 coef  
#install.packages("tidyverse")  
library(tidyverse)  
library(pROC)  
library(fhpredict)  
library(tidyverse)  
  
library(kwb.flusshygiene)  
  
  
  
#if (FALSE)  
   
  
   
 #### Laden von Testdaten ###################  
   
 rivers <- c("havel")  
 river <- "havel"  
 #river\_paths <- kwb.flusshygiene::get\_paths()[paste0(rivers, "data")]  
   
 river\_paths <- list(havel = "Y:/SUW\_Department/Projects/FLUSSHYGIENE/Data-Work packages/Daten/Daten\_TestPackage\_Berlin/Havel/DATA\_preprocessed\_csv")  
   
 river\_paths <- list(havel = "/Users/heiko.langer/Masterarbeit\_lokal/Data\_preprocess/Daten\_TestPackage\_Berlin/Havel/DATA\_preprocessed\_csv")  
   
 river\_data <- lapply(river\_paths, kwb.flusshygiene::import\_riverdata)  
   
 river <- "havel"  
   
 names(river\_data) <- rivers  
  
   
#  
 calc\_t <- function (datalist=river\_data$havel, onlysummer) {  
 #heiko  
 #datalist<- river\_data1$havel  
 phy\_data <- datalist[-1] # Entfernung der Hygienedaten  
   
 if(onlysummer==T){  
 hyg\_df <- subset(datalist[[1]],  
 subset = lubridate::month(datum) %in% 5:9) # Filtern nach Sommer, warum hier 5:9 und beim anderen 4:9?  
   
 data\_summer <- lapply(phy\_data, function(df){  
   
 df <- subset(df, subset = lubridate::month(datum) %in% 4:9)   
   
 }  
 )  
 }   
   
   
 # z\_standardize <- function (x) {  
   
 # y = (x - mean(x, na.rm=T))/sd(x, na.rm=T)  
   
 # }  
   
 log\_transorm\_rain <- function(df) { #log transforming rain data  
   
 for (site in names(df)[-1]) { # every col gets treatment  
   
 df2 <- subset(df, select = c("datum", site))  
   
 if (grepl("^r\_.\*",site)) { # rain gets log-transformed and 1/sigma2  
   
 df2[[site]] <- log(df2[[site]]+1)  
   
 # df2[[site]] <- df2[[site]]/sd(df2[[site]], na.rm=T)  
   
 } #else {  
   
 # df[[site]] <- z\_standardize(df2[[site]]) # standardize  
   
 # }  
   
 df[[site]] <- df2[[site]]  
   
 }  
   
 return(df)  
   
 }  
   
 data\_t <- lapply(data\_summer, log\_transorm\_rain)  
   
 result <- append(list(hyg\_df), data\_t)  
   
 names(result) <- names(datalist)  
   
 return(result)  
   
 }  
 ### Anwenden von calc\_t auf Inputliste  
   
 river\_data\_ts <- lapply(river\_data, function(river\_list){  
   
 river\_ts <- calc\_t(river\_list, onlysummer = T) # use function  
   
 add\_meancol <- function (df) { # for rain and i #edit: + ka #2ndedit: + q  
   
 prefix <- unique(sub("([a-z])\_.\*","\\1",names(df)[-1]))  
   
 for (pre in prefix) {  
   
 df2 <- dplyr::select(df, dplyr::starts\_with(pre))  
   
 df[,paste0(pre,"\_mean")] <- rowMeans(df2, na.rm=T)  
   
 }  
   
   
   
 return(df)  
   
 }  
   
 add\_sumcol <- function (df) { # originally for ka, but not used  
   
 prefix <- unique(sub("([a-z])\_.\*","\\1",names(df)[-1]))  
   
 if (length(df) > 2)  
   
 df[,paste0(prefix,"\_sum")] <- rowSums(df[,-1], na.rm=T)  
   
 return(df)  
   
 }  
   
   
   
 q\_pos <- grep("^q", names(river\_ts)[-1])+1  
   
   
 if (length(q\_pos) == 1)  
   
 river\_ts[[q\_pos]] <- add\_meancol(river\_ts[[q\_pos]])  
   
 ka\_pos <- grep("^ka", names(river\_ts)[-1])+1  
   
 if (length(ka\_pos) == 1)  
   
 river\_ts[[ka\_pos]] <- add\_meancol(river\_ts[[ka\_pos]])  
   
 i\_pos <- grep("^i", names(river\_ts)[-1])+1  
   
 if (length(i\_pos) == 1)  
   
 river\_ts[[i\_pos]] <- add\_meancol(river\_ts[[i\_pos]])  
   
 r\_pos <- grep("^r", names(river\_ts)[-1])+1  
   
 river\_ts[[r\_pos]] <- add\_meancol(river\_ts[[r\_pos]])  
   
 return(river\_ts)  
   
 })  
   
   
   
 rm(river\_data,calc\_t)  
   
 river = "havel"  
 pattern = "(i\_mean|q\_mean\_mean|r\_mean\_mean|ka\_mean\_mean)"  
 riverdata <- river\_data\_ts[[river]]  
   
 # prepare variables out of all cominations (given by pattern)  
   
 # variables for interaction get replaced by q\_new (remove q\_old)  
   
 vars1 <- (riverdata[-1] %>% unroll\_physical\_data() %>%  
   
 lapply(names) %>% unlist() %>% unique())[-1]  
   
 vars2 <- vars1[stringr::str\_detect(vars1, pattern)]  
   
   
   
 # prepare formulas  
   
 data <- process\_model\_riverdata(riverdata, c("log\_e.coli", vars2)) %>%  
   
 dplyr::select(-datum)  
   
   
   
   
   
 data <- na.omit(data)  
   
 data <- data %>% filter(log\_e.coli > log10(15)) #why-heiko?  
   
   
 #Definition of models  
   
 # Definition of null and full models  
 #stepwise models  
 null <- lm(log\_e.coli ~ 1, data = data) #model with only 1 variable  
   
 full <- lm(log\_e.coli ~ .^2, data = data)  
   
 #heiko models  
   
   
}   
 {  
 #heiko  
 {  
   
   
 get\_coef\_1se\_cv <- function(df){  
 tmp\_coeffs <- coef(df, s = "lambda.1se")  
 a <- data.frame(name = tmp\_coeffs@Dimnames[[1]][tmp\_coeffs@i + 1], coefficient = tmp\_coeffs@x)  
 return(a)  
 }  
 get\_coef\_min\_cv <- function(df){  
 tmp\_coeffs <- coef(df, s = "lambda.min")  
 a <- data.frame(name = tmp\_coeffs@Dimnames[[1]][tmp\_coeffs@i + 1], coefficient = tmp\_coeffs@x)  
 return(a)  
 }  
   
   
   
 get\_coef\_fixed\_lambda <- function(df,lambda){  
 tmp\_coeffs <- coef(df, s = lambda)  
 a <- data.frame(name = tmp\_coeffs@Dimnames[[1]][tmp\_coeffs@i + 1], coefficient = tmp\_coeffs@x)  
 return(a)  
 }  
   
   
 }  
 get\_formula\_variable\_names <- function(formula\_a,df){   
 mf <- model.frame(formula\_a, data=df)  
 mt <- attr(mf, "terms")  
 predvarnames <- attr(mt, "term.labels")  
 predvarnames  
 }  
   
 #lasso  
 #build/integrate here into folds to train with same cross validation  
 #fold1<-train\_rows[[1]]  
   
 #training\_heiko<-data[fold1,]  
   
 part1<-names(data)[1]  
 form<-formula(paste(part1," ~ (.)^2"))  
 get\_formula\_variable\_names(form,data)  
 #training\_heiko\_features <- (training\_heiko%>% select(-log\_e.coli))  
 #sparse.model.matrix(form, training\_heiko)  
   
 #form <- log\_e.coli ~ (.)ˆ2  
 #training\_heiko\_features\_matrix <- (data.frame.2.sparseMatrix(training\_heiko\_features))  
 train\_sparse <- sparse.model.matrix(form, data) #data must be dataframe  
 #train\_sparse <- sparse.model.matrix(training\_heiko$log\_e.coli~(.)ˆ2, training\_heiko[,3:ncol(training\_heiko)]) #data must be dataframe  
 #form <- Y ~ (x + y + z)^2  
 #testing\_heiko<-data[-fold1,]  
   
 # test\_sparse <- sparse.model.matrix(testing\_heiko$log\_e.coli~., testing\_heiko[,3:ncol(testing\_heiko)]) #data must be dataframe  
 set.seed(4)  
  
 {   
 fit\_lasso\_base <- glmnet(train\_sparse, data$log\_e.coli , na.rm =T, standardize = F, alpha = 1,relax = F)  
   
 fit\_lasso\_base\_cross <- cv.glmnet(train\_sparse, data$log\_e.coli,type.measure="mse", alpha=1, family="gaussian", nfolds = 3,standardize = F,relax = F)#--> alpha =1: lasso regressio  
   
   
 fit\_lasso\_base\_stand <- glmnet(train\_sparse, data$log\_e.coli , na.rm =T, standardize = T, alpha = 1,relax = F)  
 fit\_lasso\_base\_cross\_stand <- cv.glmnet(train\_sparse, data$log\_e.coli,type.measure="mse", alpha=1, family="gaussian", nfolds = 3,standardize = T,relax = F)#--> alpha =1: lasso regressio  
   
 #par(mfrow=c(2,2))  
 #plot(fit\_lasso\_base, xvar="lambda", label = T, main = "lasso\_base")  
 #plot(fit\_lasso\_base\_cross,main="LASSO")  
   
 #plot(fit\_lasso\_base\_stand, xvar="lambda", label = T, main = "lasso\_base\_stand")  
 #plot(fit\_lasso\_base\_cross\_stand,main="LASSO")  
   
 #plot(fit\_elnet\_base, xvar="lambda", label = T, main = "elnet\_base")  
 #plot(fit\_elnet\_base\_cross,main="elnet")  
   
 #plot(fit\_elnet\_base\_stand, xvar="lambda", label = T, main = "elnet\_base\_stand")  
 #plot(fit\_elnet\_base\_cross\_stand,main="elnet")  
   
   
   
 get\_feature\_selection\_coeficient\_names\_as\_formular\_1se <- function(algorithm\_list){  
 #fit\_lasso\_base\_cross  
 #algorithm\_list<-fit\_lasso\_base\_cross  
 coef\_1se<- get\_coef\_1se\_cv(algorithm\_list)  
 if(dim(coef\_1se)[1]==1){  
 print("only intercept. nothing to model")  
 }else{  
 coef\_name\_lambda\_1se<-coef\_1se$name[-1]  
 #a<-str("")  
 coefficients<-paste(coef\_name\_lambda\_1se, collapse = " + " )  
   
 formel<-paste("log\_e.coli ~ ", coefficients)  
 formel  
 formula\_from\_selector<-formula(formel)  
 }  
 return(formula\_from\_selector)  
 }  
 get\_feature\_selection\_coeficient\_names\_as\_formular\_lambda\_min <- function(algorithm\_list){  
 #algorithm\_list<-fit\_lasso\_base\_cross  
 coef\_lambda\_min<- get\_coef\_min\_cv(algorithm\_list)  
 coef\_name\_lambda\_min<-coef\_lambda\_min$name[-1]  
 #a<-str("")  
 coefficients<-paste(coef\_name\_lambda\_min, collapse = " + " )  
 formel<-paste("log\_e.coli ~ ", coefficients)  
 formula\_from\_selector<-formula(formel)  
 return(formula\_from\_selector)  
 }   
   
 coef\_1se\_fit\_lasso\_base\_cross<-get\_coef\_1se\_cv (fit\_lasso\_base\_cross)  
 coef\_1se\_fit\_lasso\_base\_cross\_stand<-get\_coef\_1se\_cv (fit\_lasso\_base\_cross\_stand)  
 coef\_lambda\_min\_fit\_lasso\_base\_cross<-get\_coef\_min\_cv (fit\_lasso\_base\_cross)  
 coef\_lambda\_min\_fit\_lasso\_base\_cross\_stand<-get\_coef\_min\_cv (fit\_lasso\_base\_cross\_stand)  
   
 # add\_new\_formulas\_to\_list\_if\_exists <- function(coef\_list){  
   
# if(exists("coef\_1se\_fit\_lasso\_base\_cross")== TRUE){  
 # idx <- length(list\_lasso)  
 # idx <- idx+1  
 # list\_lasso[[idx]] <-coef\_1se\_fit\_lasso\_base\_cross  
 # }  
 #}  
   
 list\_lasso <- list()  
   
 coef\_1se\_fit\_lasso\_base\_cross <-get\_feature\_selection\_coeficient\_names\_as\_formular\_1se(fit\_lasso\_base\_cross)  
   
 if(exists("coef\_1se\_fit\_lasso\_base\_cross")== TRUE){  
 idx <- length(list\_lasso)  
 idx <- idx+1  
 list\_lasso[[idx]] <-coef\_1se\_fit\_lasso\_base\_cross  
 }  
 coef\_1se\_fit\_lasso\_base\_cross\_stand <-get\_feature\_selection\_coeficient\_names\_as\_formular\_1se(fit\_lasso\_base\_cross\_stand)  
 if(exists("coef\_1se\_fit\_lasso\_base\_cross\_stand")== TRUE){  
 idx <- length(list\_lasso)  
 idx <- idx+1  
 list\_lasso[[idx]] <-coef\_1se\_fit\_lasso\_base\_cross\_stand  
 }  
   
 coef\_lambda\_min\_fit\_lasso\_base\_cross <-get\_feature\_selection\_coeficient\_names\_as\_formular\_lambda\_min(fit\_lasso\_base\_cross)  
 if(exists("coef\_lambda\_min\_fit\_lasso\_base\_cross")== TRUE){  
 idx <- length(list\_lasso)  
 idx <- idx+1  
 list\_lasso[[idx]] <-coef\_lambda\_min\_fit\_lasso\_base\_cross  
 }  
   
 coef\_lambda\_min\_fit\_lasso\_base\_cross\_stand <-get\_feature\_selection\_coeficient\_names\_as\_formular\_lambda\_min(fit\_lasso\_base\_cross\_stand)  
 if(exists("coef\_lambda\_min\_fit\_lasso\_base\_cross\_stand")== TRUE){  
 idx <- length(list\_lasso)  
 idx <- idx+1  
 list\_lasso[[idx]] <-coef\_lambda\_min\_fit\_lasso\_base\_cross\_stand  
 }  
   
 #check if all 4 coefficients exist and remove intercepts  
 idx <-0  
 for(element in list\_lasso){  
 idx<-idx+1  
 if(typeof(element)!="language"){  
 list\_lasso <- list\_lasso[-idx]  
 print("f")  
 }  
 }  
 list\_lasso  
 #print(paste(length(list\_lasso)," new models added"))   
 model\_lsit<-list()  
 #list\_lasso  
 #builded linear model  
 heiko\_lm\_1<-lm(list\_lasso[[1]], data = data)  
 heiko\_lm\_2<-lm(list\_lasso[[2]],data=data)  
 heiko\_lm\_3<-lm(list\_lasso[[3]],data=data)  
 heiko\_lm\_4<-lm(list\_lasso[[4]],data=data)  
   
 list\_heiko\_lm <- list()  
 list\_heiko\_lm[[1]]<- heiko\_lm\_1  
 list\_heiko\_lm[[2]]<- heiko\_lm\_2  
 list\_heiko\_lm[[3]]<- heiko\_lm\_3  
 list\_heiko\_lm[[4]]<- heiko\_lm\_4  
 #for(form in list\_lasso){  
   
 # heiko\_lm <- lm(form, data = data)  
 # heiko\_lm<-list(heiko\_lm)  
 # append(heiko\_lm,model\_lsit)  
 # }  
 #heiko\_lm<- lm(formula\_heiko\_1, data = data)   
   
#nicht mehr benötigt   
 #### Anwenden der Hauptfunktion ###################  
   
 stepwise <- function (river, pattern, data, null, full ){  
 # Definition maximum number of steps  
   
 nsteps <- 5 #ifelse(round(nrow(data)/10) < 10, round(nrow(data)/10), 5 )  
   
 selection <- list()  
   
 fmla <- list()  
   
   
   
 # Creating list of candidate models with 1 ...n predictors   
 #split up this piece in stpe and new algorithms/formulars  
   
 for(i in 1: nsteps){  
   
   
   
 selection[[i]] <- step(null, data = data,  
   
 direction = "forward",  
   
 list(lower=null, upper=full), steps = i, trace=FALSE)   
   
   
 fmla[[i]] <- as.list(selection[[i]]$call)$formula  
   
   
   
 }  
   
 #heiko\_add\_formular to fmla list function function  
 #selection[[6]] <- heiko\_lm  
 #fmla[[6]] <- as.list(selection[[6]]$call)$formula  
 step\_returns <- list(fmla, selection)  
 return(step\_returns)  
   
 }  
   
   
   
   
   
 # order of pattern, q\_old and q\_new is important!  
   
 #fb <- stepwise(river = river, pattern = "(i\_mean|q\_mean\_mean|r\_mean\_mean|ka\_mean\_mean)", data,null, full)#,  
 step\_returns <- stepwise(river = river, pattern = "(i\_mean|q\_mean\_mean|r\_mean\_mean|ka\_mean\_mean)", data,null, full)  
 fmla <- step\_returns[[1]]  
 selection <- step\_returns[[2]]  
   
   
   
   
   
   
#adding new linear models, featureselection with lasso/elnet  
#selection<-append(selection, list(heiko\_lm\_1,heiko\_lm\_2,heiko\_lm\_3,heiko\_lm\_4))  
 selection<-append(selection, list\_heiko\_lm)  
 fb<- selection   
#fb[6] <- list(heiko\_lm)  
   
 #selection[6] <- list(heiko\_lm)  
 #selection  
 #fb  
   
 fmla\_heiko\_1 <-eval(heiko\_lm\_1$call$formula)  
 fmla\_heiko\_2 <-eval(heiko\_lm\_2$call$formula)  
 fmla\_heiko\_3 <-eval(heiko\_lm\_3$call$formula)  
 fmla\_heiko\_4 <-eval(heiko\_lm\_4$call$formula)  
   
 fmla\_heiko <- list()  
 fmla\_heiko[[1]]<- fmla\_heiko\_1  
 fmla\_heiko[[2]]<- fmla\_heiko\_2  
 fmla\_heiko[[3]]<- fmla\_heiko\_3  
 fmla\_heiko[[4]]<- fmla\_heiko\_4  
 # as.list(selection[[6]]$call)$formula  
   
 fmla<-append(fmla, fmla\_heiko)  
 #fmla  
 if(class(fmla[[length(fmla)]]) !="formula"){  
 print("new element is no formula!!")  
 }  
   
   
 #add my models here  
   
   
 #q\_old = "q\_cochem",  
   
 #q\_new = "q\_cochem\_abs\_1")  
   
  
  
   
   
   
 names(fb) <- sprintf(paste0(river,"model\_%02d"), seq\_along(1:length(fb)))  
   
   
   
 ################ Validation ########################  
   
   
   
 # calculate statistical tests for residuals: Normality and s2 = const  
   
 # shapiro-wilk test and breusch-pagan test  
   
 get\_stat\_tests <- function(model) {  
 c(N = shapiro.test(model$residuals)$p.value, lmtest::bptest(model)$p.value,  
 R2 = summary(model)[["adj.r.squared"]], n\_obs = length(model$residuals))  
   
 }  
   
   
   
 # Eliminieren von modelled die doppelt vorkommen, da forward selection früher  
   
 #fertig als n steps  
   
 #heiko add fb beforehand to this  
 #fb  
 unique\_index <- length(unique(fb))  
 fb <- fb[1:unique\_index]  
   
   
   
 # testing for classical statistical model assumtions, normality of residuals and  
   
 # heteroskelasdicity  
 river\_stat\_tests <- sapply(fb, get\_stat\_tests)%>%  
 t() %>%  
 dplyr::as\_tibble(rownames = "model") %>%  
 dplyr::bind\_rows(.id = "river") %>%  
 dplyr::mutate(stat\_correct = N > .05 & BP > .05)  
   
   
   
 # creating list of independent training rows  
 #-test/train split  
   
 #weirde zeile, setze alle stat tests auf 0  
 river\_stat\_tests$in95 <- river\_stat\_tests$below95 <-river\_stat\_tests$below90 <- river\_stat\_tests$in50 <- 0  
   
   
 train\_rows <- caret::createFolds(1:nrow(fb[[paste0(river, "model\_01")]]$model),  
   
 k = 5, list = T, returnTrain = T)  
   
  
   
 if(class(fmla[[length(fmla)]]) !="formula"){  
 print("new element is no formula!!")  
 }  
   
 test\_beta <- function(true, false, percentile){  
 if( pbeta(q = percentile, shape1 = true + 1, shape2 = false + 1) > 0.05){  
 TRUE}  
 else{FALSE}  
   
 }  
   
   
 names(fmla) <- sprintf(paste0(river,"model\_%02d"), seq\_along(1:length(fb)))  
   
  
   
   
 counter<-0  
   
 #fb<-fb[-6]  
 #names(fb)  
}  
  
 for(i in names(fb)){  
 counter<- counter+1  
 #i="havelmodel\_01"  
 for(j in 1:5){  
 counter <- counter+1  
 # j=1  
   
   
   
   
 training <- as.data.frame(fb[[i]]$model)[c(train\_rows[[j]]),]  
 #training <- as.data.frame(fb[[6]]$model)[c(train\_rows[[1]]),]  
 test <- as.data.frame(fb[[i]]$model)[-c(train\_rows[[j]]),]  
 #test <- as.data.frame(fb[[6]]$model)[-c(train\_rows[[1]]),]  
 #formel<-formula(formula\_heiko\_1)  
   
   
 #fmla[6]<- list(formel)  
   
   
   
 fit <- rstanarm::stan\_glm(fmla[[i]], data = training, refresh=0) #fitting #suppress print out with refresh =0  
 #fit <- rstanarm::stan\_glm(fmla[[1]], data = training) #fitting  
   
   
 df <- apply(rstanarm::posterior\_predict(fit, newdata = test), 2, quantile, #predicting  
   
 probs = c(0.025, 0.25, 0.75, 0.9, 0.95, 0.975)) %>% t() %>% as.data.frame() %>%  
   
 dplyr::mutate(log\_e.coli = test$log\_e.coli, #evaluating ther model has to be classified correctly with every single test train split  
 #--> here 5 different splits, if all validations correct than everywhere ==5  
   
 below95 = log\_e.coli < `95%`,  
   
 below90 = log\_e.coli < `90%`,  
   
 within95 = log\_e.coli < `97.5%`& log\_e.coli > `2.5%`,  
   
 within50 = log\_e.coli < `75%`& log\_e.coli > `25%`,  
   
 )  
   
 #validation step if allpercentile categories are set to 1  
   
 river\_stat\_tests$in95[river\_stat\_tests$model == i] <-  
   
 river\_stat\_tests$in95[river\_stat\_tests$model == i] +  
   
 test\_beta(true = sum(df$within95), false = sum(!df$within95), percentile = .95 )  
   
 river\_stat\_tests$below95[river\_stat\_tests$model == i] <-  
   
 river\_stat\_tests$below95[river\_stat\_tests$model == i] +  
   
 test\_beta(true = sum(df$below95), false = sum(!df$below95), percentile = .95 )  
   
 river\_stat\_tests$below90[river\_stat\_tests$model == i] <-  
   
 river\_stat\_tests$below90[river\_stat\_tests$model == i] +  
   
 test\_beta(true = sum(df$below90), false = sum(!df$below90), percentile = .90 )  
   
 river\_stat\_tests$in50[river\_stat\_tests$model == i] <-  
   
 river\_stat\_tests$in50[river\_stat\_tests$model == i] +  
   
 test\_beta(true = sum(df$within50), false = sum(!df$within50), .5)  
   
   
   
 }   
   
 }   
   
#fmla  
  
}   
   
}

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

## Loading required package: Matrix

## Loaded glmnet 4.0

##   
## Attaching package: 'purrr'

## The following object is masked from 'package:magrittr':  
##   
## set\_names

## ── Attaching packages ───────────────────────────────────────────────────────────────────────────────────────────── tidyverse 1.3.0 ──

## ✓ ggplot2 3.3.1 ✓ readr 1.3.1  
## ✓ tibble 3.0.1 ✓ stringr 1.4.0  
## ✓ tidyr 1.1.0 ✓ forcats 0.5.0

## ── Conflicts ──────────────────────────────────────────────────────────────────────────────────────────────── tidyverse\_conflicts() ──  
## x tidyr::expand() masks Matrix::expand()  
## x tidyr::extract() masks magrittr::extract()  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()  
## x tidyr::pack() masks Matrix::pack()  
## x purrr::set\_names() masks magrittr::set\_names()  
## x tidyr::unpack() masks Matrix::unpack()

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

##   
## Attaching package: 'kwb.flusshygiene'

## The following objects are masked from 'package:fhpredict':  
##   
## build\_model, predict\_quality

sorted\_modellist <- river\_stat\_tests %>%  
   
 filter( below95 == 5 & below90 == 5& in95) %>%  
   
 dplyr::arrange(desc(in50), desc(R2))  
   
 #river\_stat\_tests  
 #sorted\_modellist  
   
 best\_valid\_model\_stats <- sorted\_modellist[1,]  
   
 best\_valid\_model <- fb[[best\_valid\_model\_stats$model]]  
   
 #coef(best\_valid\_model)  
   
   
 #refit best model  
 #stanfit <- rstanarm::stan\_glm(fmla[[best\_valid\_model\_stats$model]],  
   
 # data = best\_valid\_model$model, refresh=0)  
   
 #brmsfit <- brms::brm(fmla[[best\_valid\_model\_stats$model]],  
   
 # data = best\_valid\_model$model, iter = 10000)  
   
   
   
 #return(list(sorted\_modellist = sorted\_modellist,  
   
 # best\_model = best\_valid\_model,  
   
 # stanfit = stanfit,  
   
 # brmsfit = brmsfit))

The used data is from river ‘Havel’: “/Data\_preprocess/Daten\_TestPackage\_Berlin/Havel/DATA\_preprocessed\_csv”

It is taken from the data folder provided by Wolfgang and has the following structure

glimpse(data)

## Rows: 36  
## Columns: 31  
## $ log\_e.coli <dbl> 2.100371, 2.428135, 2.252853, 2.838849, 1.662758, …  
## $ ka\_mean\_mean\_12 <dbl> 38.0145, 69.7390, 0.0000, 109.7215, 32.4950, 36.65…  
## $ ka\_mean\_mean\_123 <dbl> 25.34300000, 108.61533333, 0.00000000, 85.48100000…  
## $ ka\_mean\_mean\_1234 <dbl> 19.00725, 86.94825, 0.00000, 97.09825, 19.73725, 2…  
## $ ka\_mean\_mean\_12345 <dbl> 15.2058, 79.3838, 13.3450, 84.9316, 17.4996, 18.48…  
## $ ka\_mean\_mean\_2345 <dbl> 17.36950, 70.51750, 16.68125, 74.91325, 5.62700, 1…  
## $ ka\_mean\_mean\_345 <dbl> 0.000000, 85.813667, 22.241667, 68.405000, 7.50266…  
## $ ka\_mean\_mean\_45 <dbl> 0.0000, 35.5365, 33.3625, 84.1075, 11.2540, 9.5690…  
## $ ka\_mean\_mean\_234 <dbl> 23.1593333, 77.6480000, 0.0000000, 87.7960000, 4.6…  
## $ ka\_mean\_mean\_23 <dbl> 34.7390, 105.4985, 0.0000, 65.7190, 0.0000, 28.857…  
## $ ka\_mean\_mean\_34 <dbl> 0.0000, 104.1575, 0.0000, 84.4750, 6.9795, 5.3530,…  
## $ q\_mean\_mean\_12 <dbl> 61.65, 75.15, 68.30, 80.30, 66.95, 25.85, 37.25, 3…  
## $ q\_mean\_mean\_123 <dbl> 61.43333, 75.16667, 70.56667, 78.16667, 65.93333, …  
## $ q\_mean\_mean\_1234 <dbl> 62.175, 68.925, 72.650, 78.325, 65.125, 27.325, 30…  
## $ q\_mean\_mean\_12345 <dbl> 61.48, 64.44, 74.76, 76.68, 64.82, 25.50, 27.00, 3…  
## $ q\_mean\_mean\_2345 <dbl> 61.325, 61.900, 76.525, 74.650, 63.975, 25.625, 23…  
## $ q\_mean\_mean\_345 <dbl> 61.36667, 57.30000, 79.06667, 74.26667, 63.40000, …  
## $ q\_mean\_mean\_45 <dbl> 61.55, 48.35, 81.05, 74.45, 63.15, 25.35, 16.85, 3…  
## $ q\_mean\_mean\_234 <dbl> 62.20000, 67.03333, 74.30000, 76.16667, 64.10000, …  
## $ q\_mean\_mean\_23 <dbl> 61.10, 75.45, 72.00, 74.85, 64.80, 25.90, 29.70, 4…  
## $ q\_mean\_mean\_34 <dbl> 62.70, 62.70, 77.00, 76.35, 63.30, 28.80, 23.05, 4…  
## $ r\_mean\_mean\_12 <dbl> 1.15234521, 1.34171239, 0.04126964, 1.93422257, 0.…  
## $ r\_mean\_mean\_123 <dbl> 0.82315166, 1.95004080, 0.06128739, 1.41172257, 0.…  
## $ r\_mean\_mean\_1234 <dbl> 0.6435257, 2.1261437, 0.3237925, 1.6186162, 0.5993…  
## $ r\_mean\_mean\_12345 <dbl> 0.5262331, 1.7113038, 0.5347389, 1.4190294, 0.5910…  
## $ r\_mean\_mean\_2345 <dbl> 0.2533034, 1.6019403, 0.6573695, 1.1881228, 0.4187…  
## $ r\_mean\_mean\_345 <dbl> 0.1088251, 1.9576981, 0.8637185, 1.0755673, 0.5476…  
## $ r\_mean\_mean\_45 <dbl> 0.08085530, 1.35319831, 1.24491623, 1.42998969, 0.…  
## $ r\_mean\_mean\_234 <dbl> 0.3187169, 2.1186057, 0.4169844, 1.3772697, 0.3723…  
## $ r\_mean\_mean\_23 <dbl> 0.42575152, 1.85068227, 0.06982273, 0.94625589, 0.…  
## $ r\_mean\_mean\_34 <dbl> 0.1347061, 2.9105750, 0.6063153, 1.3030099, 0.5425…

The prebuild list of 31 averaged variables/features are increased through interactions between each other. This is done by building all interaction formulars with “formula(paste(part1,” ~ (.)^2“)” So every feature besides e.coli will be multiplied with every other feature which leads to 466 features from which the Lasso-algorithms are selecting the most important.

here is an insight of the last 50 formulas

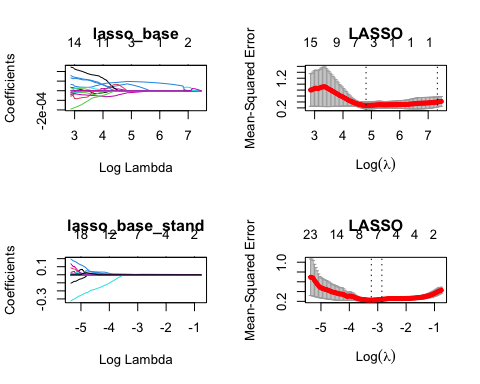
tail(get\_formula\_variable\_names(form,data), 50)

## [1] "q\_mean\_mean\_34:r\_mean\_mean\_345" "q\_mean\_mean\_34:r\_mean\_mean\_45"   
## [3] "q\_mean\_mean\_34:r\_mean\_mean\_234" "q\_mean\_mean\_34:r\_mean\_mean\_23"   
## [5] "q\_mean\_mean\_34:r\_mean\_mean\_34" "r\_mean\_mean\_12:r\_mean\_mean\_123"   
## [7] "r\_mean\_mean\_12:r\_mean\_mean\_1234" "r\_mean\_mean\_12:r\_mean\_mean\_12345"   
## [9] "r\_mean\_mean\_12:r\_mean\_mean\_2345" "r\_mean\_mean\_12:r\_mean\_mean\_345"   
## [11] "r\_mean\_mean\_12:r\_mean\_mean\_45" "r\_mean\_mean\_12:r\_mean\_mean\_234"   
## [13] "r\_mean\_mean\_12:r\_mean\_mean\_23" "r\_mean\_mean\_12:r\_mean\_mean\_34"   
## [15] "r\_mean\_mean\_123:r\_mean\_mean\_1234" "r\_mean\_mean\_123:r\_mean\_mean\_12345"   
## [17] "r\_mean\_mean\_123:r\_mean\_mean\_2345" "r\_mean\_mean\_123:r\_mean\_mean\_345"   
## [19] "r\_mean\_mean\_123:r\_mean\_mean\_45" "r\_mean\_mean\_123:r\_mean\_mean\_234"   
## [21] "r\_mean\_mean\_123:r\_mean\_mean\_23" "r\_mean\_mean\_123:r\_mean\_mean\_34"   
## [23] "r\_mean\_mean\_1234:r\_mean\_mean\_12345" "r\_mean\_mean\_1234:r\_mean\_mean\_2345"   
## [25] "r\_mean\_mean\_1234:r\_mean\_mean\_345" "r\_mean\_mean\_1234:r\_mean\_mean\_45"   
## [27] "r\_mean\_mean\_1234:r\_mean\_mean\_234" "r\_mean\_mean\_1234:r\_mean\_mean\_23"   
## [29] "r\_mean\_mean\_1234:r\_mean\_mean\_34" "r\_mean\_mean\_12345:r\_mean\_mean\_2345"  
## [31] "r\_mean\_mean\_12345:r\_mean\_mean\_345" "r\_mean\_mean\_12345:r\_mean\_mean\_45"   
## [33] "r\_mean\_mean\_12345:r\_mean\_mean\_234" "r\_mean\_mean\_12345:r\_mean\_mean\_23"   
## [35] "r\_mean\_mean\_12345:r\_mean\_mean\_34" "r\_mean\_mean\_2345:r\_mean\_mean\_345"   
## [37] "r\_mean\_mean\_2345:r\_mean\_mean\_45" "r\_mean\_mean\_2345:r\_mean\_mean\_234"   
## [39] "r\_mean\_mean\_2345:r\_mean\_mean\_23" "r\_mean\_mean\_2345:r\_mean\_mean\_34"   
## [41] "r\_mean\_mean\_345:r\_mean\_mean\_45" "r\_mean\_mean\_345:r\_mean\_mean\_234"   
## [43] "r\_mean\_mean\_345:r\_mean\_mean\_23" "r\_mean\_mean\_345:r\_mean\_mean\_34"   
## [45] "r\_mean\_mean\_45:r\_mean\_mean\_234" "r\_mean\_mean\_45:r\_mean\_mean\_23"   
## [47] "r\_mean\_mean\_45:r\_mean\_mean\_34" "r\_mean\_mean\_234:r\_mean\_mean\_23"   
## [49] "r\_mean\_mean\_234:r\_mean\_mean\_34" "r\_mean\_mean\_23:r\_mean\_mean\_34"

The features are getting selected through the lasso algorithm with cross validation without (1.row) or with (2.row) internal standardization of the features

The following 4 graphs show the the number of selected features (upper sclae) dependend on log lambda (lower scale) The left dotted line shows the lambda with the minimal MSE - identified throug cross-validation The right dotted line shows the lambda 1 standard deviation away from the with the minimal MSE - identified throug cross-validation (shall give a better generalization than minimum mse)

par(mfrow=c(2,2))  
 plot(fit\_lasso\_base, xvar="lambda", label = T, main = "lasso\_base")  
 plot(fit\_lasso\_base\_cross,main="LASSO")  
   
 plot(fit\_lasso\_base\_stand, xvar="lambda", label = T, main = "lasso\_base\_stand")  
 plot(fit\_lasso\_base\_cross\_stand,main="LASSO")

 lambda 1se for lasso base cross

get\_coef\_1se\_cv(fit\_lasso\_base\_cross)$name

## [1] "(Intercept)" "q\_mean\_mean\_123:q\_mean\_mean\_23"

lambda 1se for lasso base cross standardized

get\_coef\_1se\_cv(fit\_lasso\_base\_cross\_stand)$name

## [1] "(Intercept)" "q\_mean\_mean\_2345"   
## [3] "ka\_mean\_mean\_12:ka\_mean\_mean\_123" "ka\_mean\_mean\_12:ka\_mean\_mean\_34"   
## [5] "q\_mean\_mean\_12:r\_mean\_mean\_12" "q\_mean\_mean\_45:r\_mean\_mean\_12"   
## [7] "q\_mean\_mean\_45:r\_mean\_mean\_12345" "q\_mean\_mean\_45:r\_mean\_mean\_45"

9 different models were build in the very first code snippet: - 5 from stepwise regression, with 1-5 explanatory variables (choosen with Information Criterion) - 4 Lasso based models:

from the two cross validated lasso fits: fit\_lasso\_base\_cross and fit\_lasso\_base\_cross\_stand (with internal standardization of features) I used lambda min and lambda-1se to get 4 lasso based models with the following modelnumbers: 06: fit\_lasso\_base\_cross lambda-1se 07: fit\_lasso\_base\_cross\_stand lambda-1se 08: fit\_lasso\_base\_cross lambda-min 09: fit\_lasso\_base\_cross\_stand lambda-min

The 4 Lasso based models were than used in the posterior function for validation. If in50, below90, below95, and in 95 are all == 5 –> the model is validated through the validation method from wolfgang.

The best Model was chosen by further checking statistical correctness and sorting R2 in descending order.

sorted\_modellist

## # A tibble: 9 x 11  
## river model N BP R2 n\_obs stat\_correct in50 below90 below95  
## <chr> <chr> <dbl> <dbl> <dbl> <dbl> <lgl> <dbl> <dbl> <dbl>  
## 1 1 have… 5.69e-1 0.155 0.697 36 TRUE 5 5 5  
## 2 1 have… 7.56e-1 0.0807 0.681 36 TRUE 5 5 5  
## 3 1 have… 8.32e-2 0.391 0.654 36 TRUE 5 5 5  
## 4 1 have… 2.22e-2 0.365 0.636 36 FALSE 5 5 5  
## 5 1 have… 5.06e-4 0.514 0.591 36 FALSE 5 5 5  
## 6 1 have… 4.95e-2 0.0225 0.560 36 FALSE 5 5 5  
## 7 1 have… 1.60e-2 0.0524 0.423 36 FALSE 5 5 5  
## 8 1 have… 8.46e-2 0.153 0.503 36 TRUE 4 5 5  
## 9 1 have… 3.68e-2 0.0236 0.474 36 FALSE 4 5 5  
## # … with 1 more variable: in95 <dbl>

Whats interesting is, that only the standardized models (07 & 09) were able to beat the stepwise builded models with this evaluation method.

Best Model here is havelmodel\_09 with the following coefficients:

names(coef(best\_valid\_model))

## [1] "(Intercept)" "q\_mean\_mean\_45"   
## [3] "ka\_mean\_mean\_12:ka\_mean\_mean\_123" "ka\_mean\_mean\_12:ka\_mean\_mean\_34"   
## [5] "q\_mean\_mean\_12:r\_mean\_mean\_12" "q\_mean\_mean\_45:r\_mean\_mean\_12"   
## [7] "q\_mean\_mean\_45:r\_mean\_mean\_1234" "q\_mean\_mean\_45:r\_mean\_mean\_45"